Integrating Processes in Temporal Logic*

Thomas Fuchß

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, 76128 Karlsruhe, Germany
email: fuchss@ira.uka.de Tel.: ++49 721 608 4245 Fax.: ++49 721 608 4211

August 1997

Abstract. In this paper we propose a technique to integrate process models in classical structures for quantified temporal (modal) logic. The idea is that in a temporal logic processes are ordinary syntactical objects with a specific semantical representation. Thus the structural information of processes can be captured and exploited to guide proofs. As an instance of this technique we present a quantified metric linear temporal logic of processes. We describe syntax and semantic of this logic especially with a focus on the process part. Finally we sketch a calculus, give some examples and discuss our experiences in doing proofs.
Keywords: temporal logic, modal logic, program logic, semantics, specification, verification, real-time.

1 Introduction

The growing complexity of todays soft- and hardware systems enforces more and more improved methodological support during the phases of development and maintenance.

The unambiguity of formal description languages based on fixed semantics ensures highly precise descriptions of requirements and design decisions. Furthermore formal specifications are the basis for any program verification. So formal methods can improve the protection against development errors. Consequently formal methods are especially useful in areas where soft- or hardware errors can lead to various damages caused by system failures. One such area is the field of steering and controlling technical systems, where time-critical and reactive systems are used.

The work we present is intended to build the basis for a semantic of a real-time specification language. This specification language should cover the entire development process reaching from the description of requirements and design decisions down to algorithms (executable specifications). The idea is to provide a framework for *temporal logics of processes* to adequately describe aspects of systems dealing with data structures, reactive and time-critical behavior, environmental influences, and their interaction.

* This research was sponsored by the DFG Graduiertenkolleg GRK 209/2-96.

The approach is to integrate process models in classical models for quantified temporal (modal) logics, together with *process symbols* as representatives at the syntactical level. We argue as follows: *If temporal logic is used for specifying real-time systems, and a unique frame for specification and verification is desired, then an embedding of processes in the set of temporal logic formulas is necessary.* By the explicit syntactical representation of processes the structural information located in the programs is caught, and made explicit on the syntactical level. Thus the information is preserved for guiding proofs of process properties. In various case studies (e.g. [6]) done with the KIV system, such a technique (compare [8]) has been successfully applied for sequential software systems.

In the following we present our approach to a combination of processes and temporal expressions.

Section 2 deals with the representation of processes. Instead of a state-based process description we prefer on the semantical level an observational based description by introducing a separate kind of semantical object, the "*process behavior set*". So the representation of a process is only one part of the model and not the entire model. Thus the model can be enhanced to describe further aspects of the system, e.g. data structures. Section 3 explains the kind of structures resulting from such a view of processes. In section 4 we consider as an instance of this scheme a linear temporal logic of processes. We describe syntax and semantics and discuss different modal operators. Section 5 presents a calculus for this logic based on signed formulas. In section 6 we give some examples illustrating our approach. Section 7 concludes with prospects for ongoing and future work.

2 Process Modeling

Processes or better time-critical and reactive systems are well studied objects in theoretical computer science. In the last years several different methods and technologies were developed to describe and analyze such systems. Important methods and models are for example *petri nets* [14], *event structures* [15], or *process algebra approaches* like CSS [11], CSP [9], and ACP [3]. A good survey about different models of concurrency can be found for example in [16]. In general the different models can be divided according to the point of view in *interleaving/non-interleaving* models, models with *linear-time/branching-time*, or models with an *explicit/implicit* representation of states. Figure 1 shows three different models which describe the same process. The left hand side shows a state-transition system, which is a graph where each node represents a state of the process mostly characterized by variable assignments. Each edge is a transition normally labeled by an event, which is the cause for the transitions. In the middle we see a synchronization tree, an unfold transition system. Each node can still be regarded as a state but it represents the whole history of the process. So a synchronization tree is a model which abstracts from states in the sense of variable assignments. Nevertheless it is possible to take internal events of the

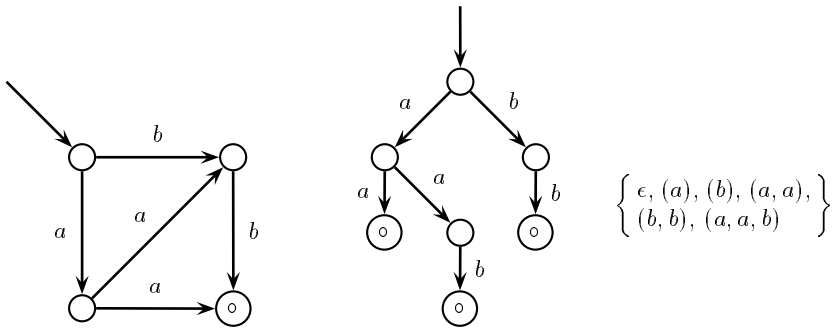

Fig. 1. A transition system, a synchronization tree, and a trace structure

system into consideration. Both *transition systems* and *synchronization trees* are used e.g. in CCS to establish the operational semantics.

The next step of abstraction leads to models without states e.g. Hoare traces [9] or Mazurkiewicz traces [10]. Such a model represents a system as a set of sequences of "observable" events. The right hand side of figure 1 shows a Hoare trace, a prefix closed set of all possible process runs.

With the above techniques it is possible to model reactive systems. However it is often necessary to take the time behavior into consideration. For a technical system it can be crucial that a set point command takes place in a fixed time interval.

A first step to obtain a solution for this task — adequate modeling of time aspects — is to enlarge the above models with real-time aspects. So we get *timed transition systems* (e.g. see [1]) or *timed observation sequences* (see [2]). Another way is to use a real-time logic (e.g. [4]). Often both are combined: the state-transition systems, synchronization trees or trace models with or without time constraints replace the classical Kripke frames as the semantical models. Such a combination is often known as a heterogeneous technique (compare [2]). It normally leads to more or less unreadable formulas without internal structure — eventually some abbreviations are used for better input. This is acceptable as long as we can stay on fully automatic tools for the analysis. But what happens if we want to regard further aspects of the system to be modeled like data structures or dependences of the process which we will not handle as arbitrary observable events. E.g., imagine a system that works properly only if the temperature of the environment is not too hot or too cold. Is this temperature an observable of the system to be modeled? Instead of an ad hoc solution it seems more promising to integrate process models in classical models of quantified temporal (modal) logics as extensions for process symbols. This has several advantages. Firstly, we get an homogeneous instead of an heterogeneous framework

for the description and analysis of real-time systems. Secondly, since a process representation is only one part in the model there is enough space to enhance such a model to describe data structures or environmental behavior. The latter can be done by defining rigid or non-rigid functions. Thirdly, the extension of a process symbol can differ from world to world. So the intension itself is non-rigid. This gives the opportunity to separate different process behavior according to environmental influences which are not immediately observable process features.

Since the state structure is given by the Kripke frame as a set of possible worlds together with an accessibility relation it is necessary to choose a model without states for the description of a single process. We have decided to take a structure comparable with Hoare traces or timed observational sequences, the "*process behavior sets*". In contrary to both, a process behavior set is neither prefix closed nor contains only finite traces. Even the information belonging to the time period of a trace is given only implicitly. The integration of this observation based process model in models of quantified temporal logic is achieved by two steps. In the first step, a single process run is defined, the *observation trace*. In the second step different observation traces are collected to build a process behavior set, which can be attached to a single possible world.

Observation traces are deterministic. So it is possible to regard them as functions from totally ordered sets to sets of observations. Since a complex process usually has more than one observable feature the observation itself can be regarded as set of "basic" observations made at the same time, where each basic observation is a a value of an observable feature. This leads to the following definition where the index set I is the set of observable features, while D is the set of values which can be observed.

Definition1 Observation Trace. Let I be an index set, $\mathcal{D} = (D_i)_{i \in I}$ a family of non-empty subsets of D, and $(T, \leq)$ a totally ordered set. Then an *observation trace* σ over $\mathcal{D}$ in T is a total function of type $T \to I \to D$ where $\sigma(t)(i) \in D_i$ for each $t \in T$ and $i \in I$.

Unless the structure of an observation trace is linear it is not necessary to restrict ourselves to linear temporal logic. A process does not influence the time structure, but the time structure can influence a process. For the following, a two sorted algebra $(T, M; \leq_t, \leq_m, d, +)$ where $(T, \leq_t)$ is a grounded partially ordered set, $(M, \leq_m)$ a totally ordered set with l.u.b. $\infty_m \in M$ and g.l.b. $0_m \in M$, d a total function of type $T \times T \to M$, and $+$ a total function of type $M \times M \to M$ satisfying the following laws for all $a, b, c \in M$ and $r, s, t \in T$:

$$
\begin{array}{ll}
(a + b) + c = a + (b + c) & a + b = b + a \\
a + 0_m = a & a + \infty_m = \infty_m \\
a + b = 0_m \Rightarrow b = 0_m & a + b = a \Rightarrow (b = 0_m \vee a = \infty_m) \\
a \leq_m b \Rightarrow \exists c\ a + c = b & 0_m \leq_m d(r, s) \leq_m \infty_m \wedge d(r, s) \neq \infty_m \\
d(r, s) = d(s, r) & r \leq_t s \leq_t t \Rightarrow d(r, t) = d(r, s) + d(s, t) \\
r \leq_t s \wedge r \neq s \Rightarrow d(r, s) \neq 0_m & r = s \Rightarrow d(r, s) = 0_m
\end{array}
$$

is used to represent a metric time structure. In such a structure $(T, \leq_t)$ represents the Kripke frame — T the possible worlds and $\leq_t$ the accessibility relation, while

d takes the time between two moments and so defines the metric. To associate a single observation trace σ to a world $t \in T$ the following should be true. If T' is the domain of σ then $(T', \leq_t)$ is a subchain of $(T, \leq_t)$ with g.l.b. $t \in T'$ such that for each $a, b \in T'$ and $c \in T$ holds $a \leq_t c \leq_t b \Rightarrow c \in T'$. Observation traces which fulfill the above conditions are for the remainder called *suitable* for the time structure at t.

Now it is easy, to get a suitable process behavior set for a time structure $TS = (T, M; \leq_t, \leq_m, d, +)$ at a possible world $t \in T$. We only have to collect different suitable observation traces for TS at t which have the same range $(I \to D)$.

Definition 2 Process Behavior Set. Let I be an index set, $\mathcal{D} = (D_i)_{i \in I}$ a family of non-empty subsets of D, $TS = (T, M; \leq_t, \leq_m, d, +)$ a metric time structure, and $t \in T$. Then a *process behavior set* P over $\mathcal{D}$ for TS at t is a subset of

$$\left\{ \sigma \middle| \begin{array}{l} \sigma \text{ is a suitable observation trace over } \mathcal{D} \\ \text{for the time structure } TS \text{ at } t. \end{array} \right\}$$

The set of all process behavior sets over $\mathcal{D}$ for TS at t is denoted by $P^{TS}_{I,\mathcal{D}}(t)$, while $P^{TS}_{I,\mathcal{D}}$ denotes $\bigcup_{t \in T} P^{TS}_{I,\mathcal{D}}(t)$.

A process behavior set is a structured object overlaying the set of possible worlds, but not one to one. We do not require that for a path in the metric time structure there is exactly one possible run (observation trace) in a process behavior set. There can also be no process run for some path or more then one for another path. If a time structure TS is used as a Kripke frame in temporal logic, a process behavior set for TS at t can be used as an extension of a process symbol in the world t. So, the representation of a process is only a part of the model and not the model itself.

3 A Linear Temporal Logic of Processes

As an instance of the above scheme we present a *quantified metric linear temporal logic* containing processes and conjunctions of processes explicitly as syntactical structures. For now we concentrate on observable features (the index set I). Instead of an unstructured set we assume that the observable features of a process can be separated in *input parameters*, *output parameters*, *input channels*, and *output channels*, since we are not interested in internal events like local variable bindings etc. The values of an observable channel represent the data exchange of the process. To avoid partiality we use a special value $\perp$ distinguished from all others to denote that no communication takes place. Each observable input parameter carries a value that initially influences the process, constant during the run. Since an output parameter is only relevant if the process is finite we will also observe $\perp$ until the process terminates. Then we observe a value different to $\perp$. To get a unique frame we have to assign to each observable feature a sort to determine the carrier set to which the values belong. So each process gets a unique type, a tuple containing four Cartesian products. One of the carrier set

for the input values followed by one for the receiving channels. Then a product of carrier sets for the channels over which the process sends data, and finally one for output values. Beside process symbols each signature contains predicate symbols and function symbols both divided in arbitrary symbols and rigid symbols the latter for the description of data structure, while non-rigid symbols are normally used to describe environmental behavior. Now we fix the signature. Each process signature $\Sigma = (S, F, P, Pid, \{\bot\})$ contains:

- S, a finite set of *Sorts*
- F, a finite set of function symbols, the disjoint union of sets $F_{\mathbf{s},s}$ and $F^r_{\mathbf{s},s}$ ($\mathbf{s} \in S^*$, $s \in S$) for $\mathbf{s} = (s_1, \ldots, s_n)$ ($n \in \mathbb{N}$) is $F_{\mathbf{s},s}$ a set of n-ary function symbols and $F^r_{\mathbf{s},s}$ a set of n-ary rigid function symbols.
- P, a finite set of predicate symbols, the disjoint union of sets $P_{\mathbf{s}}$ and $P^r_{\mathbf{s}}$ ($\mathbf{s} \in S^*$) for $\mathbf{s} = (s_1, \ldots, s_n)$ ($n \in \mathbb{N}$) is $P_{\mathbf{s}}$ a set of n-ary predicate symbols and $P^r_{\mathbf{s}}$ a set of n-ary rigid predicate symbols.
- Pid, a finite set of process symbols, the disjoint union of sets $Pid_{\mathbf{s}^i:\mathbf{s}^r:\mathbf{s}^s:\mathbf{s}^o}$ ($\mathbf{s}^k = (s^k_1, \ldots, s^k_{n_k}) \in S^*$, $n_k \in \mathbb{N}$, $k \in \{i, r, s, o\}$) of process symbols. By $(\mathbf{s}^i : \mathbf{s}^r : \mathbf{s}^s : \mathbf{s}^o)$ the different types are denoted.[2]
- $\bot$, an extra symbol denoting that no communication takes place.

For any signature we assume systems $X = (X)_{s \in S}$, $XP = (XP)_{s \in S}$, and $XC = (XC)_{s \in S}$ of countable, pairwise disjoint sets X_s of *logical variables*, XP_s of *program variables*, and XC_s of *channel variables* for each sort $s \in S$. For the logical variables we will use objectual domains while for the program and channel variables we will stay on conceptual domains. So we get a separation in rigid and non-rigid variables, too. After we have fixed the basic elements of the linear temporal logic of processes we conclude with the definition of PQ-models, structures which are based on classical Q-models for quantified model logic. A detailed introduction to quantified modal logic can be found e.g. in [7]. There, a *Q-model*, is described as follows. A Q-model (W, R, D, Q, a) contains a set W of possible worlds, a binary relation R on W, a non-empty set D of possible objects, some item Q which determines the domain of quantification, and an assignment function a, which interprets the function and predicate symbols by assigning them the corresponding kind of intension with respect to W and D. Starting with this general point of view, we give a detailed description of our modal structures. Since we have process symbols and different kinds of variables in addition, we call our structures P(rocess)Q-models. A PQ_Σ-model $\mathcal{F} = ((\mathbb{N}, \mathbb{N} \cup \{\infty\}; \leq, \leq_\infty, d, +_\infty), \mathcal{D}, \{\bot_\mathcal{F}\}, \mathcal{Q}_X, \mathcal{Q}_{XP}, \mathcal{Q}_{XC}, a)$ for a given signature Σ consists of:

$$(\mathbb{N}, \mathbb{N} \cup \{\infty\}; \leq, \leq_\infty, d, +_\infty)$$

A metric time structure where the natural numbers ($\mathbb{N}$) are the possible worlds with "less-equal" ($\leq$) as accessibility relation. For the metric we uses $\mathbb{N}$ enlarged by it's l.u.b. ∞. By $\leq_\infty$ we denote $\leq \cup \{(n, \infty) | n \in \mathbb{N} \cup \{\infty\}\}$, by d the difference

[2] The superscripts i, r, s, and o denote the different tuples for input, received values, sent values, and output.

between two natural numbers, and by $+_\infty$ the extension of the "addition" by $\{((n,\infty),\infty)|n\in\mathbb{N}\}$ and $\{((\infty,n),\infty)|n\in\mathbb{N}\}$.

$$\mathcal{D} = (D_s)_{s\in S} \text{ and } \{\bot_\mathcal{F}\}$$

A family of countable non-empty carrier sets D_s for the different sorts $s \in S$. Together with a set $\{\bot_\mathcal{F}\}$ disjoint to each D_s containing the special object $\bot_\mathcal{F}$ the extension for $\bot$.

$$\mathcal{Q}_X = (Q_{X_s})_{s\in S},\ \mathcal{Q}_{XP} = (Q_{XP_s})_{s\in S},\ \text{and } \mathcal{Q}_{XC} = (Q_{XC_s})_{s\in S}$$

Three different families of sets $Q_{X_s} = D_s$, $Q_{XP_s} = \{f|f : \mathbb{N} \to D_s\}$, and $Q_{XC_s} = \{f|f : \mathbb{N} \to \mathbb{N} \cup \{\bot_\mathcal{F}\}\}$, the domains of quantification for logical, program, and channel variables. Thereby we have a quantification equal to the modal logics $Q3$ and QC (see e.g. [7]). And finally

$$a$$

the assignment function which interprets the function, predicate, and process symbols by assigning them the corresponding kind of intension. For

functions, a is a total function, which assigns to each n-ary function symbol $f \in F_{(s_1,\ldots,s_n),s} \cup F^r_{(s_1,\ldots,s_n),s}$ a total function $a(f) : \mathbb{N} \to (D_{s_1}\times\cdots\times D_{s_n}) \to D_s$. If f is rigid then $a(f)$ is constant.

predicates, a is a total function, which assigns to each n-ary predicate symbol $p \in P_{(s_1,\ldots,s_n)} \cup P^r_{(s_1,\ldots,s_n)}$ a total function $a(p) : \mathbb{N} \to P(D_{s_1}\times\cdots\times D_{s_n})$ from $\mathbb{N}$ into the power set of $D_{s_1}\times\cdots\times D_{s_n}$. If p is rigid then $a(p)$ is constant.

processes, a is a total function, which assigns to each process symbol $q \in Pid_{\mathbf{S}^i:\mathbf{S}^r:\mathbf{S}^s:\mathbf{S}^o}$ $(\mathbf{s}^k = (s^k_1,\ldots,s^k_{n_k}) \in S^*,\ n_k \in \mathbb{N},\ k \in \{i,r,s,o\})$ a total function $a(q) : \mathbb{N} \to P^{TS}_{I,\mathcal{D}'}$ where $a(q)(n) \in P^{TS}_{I,\mathcal{D}'}(n)$ with:[3]

$$\begin{aligned} TS &= (\mathbb{N}, \mathbb{N}\cup\{\infty\}; \leq, \leq_\infty, d, +_\infty) && \text{(the time structure)} \\ I &= \biguplus_{k\in\{i,r,s,o\}} \biguplus_{m\in\{1,\ldots,n_k\}} s^k_m && \text{(the observable features)} \qquad (*) \\ D'_{s^k_m} &= \begin{cases} D_{s^k_m} & \text{if } k = i \\ D_{s^k_m} \cup \{\bot_\mathcal{F}\} & \text{else} \end{cases} && \text{(the observable values)} \end{aligned}$$

fulfilling the additional laws for all observation traces $\sigma \in a(q)(n)$ and $n \in \mathbb{N}$: According to our requirement that input values are constant is

$$\sigma(t)(s^i_m) = \sigma(t')(s^i_m) \text{ for all } t,t' \text{ in the domain of } \sigma \text{ and } m \in \{1,\ldots,n_i\}$$

For all t in the domain of σ that are less then the g.l.b. of the domain of σ if it exists is

$$\sigma(t)(s^o_m) = \bot_\mathcal{F} \text{ for all } m \in \{1,\ldots,n_o\})$$

If the g.l.b. (let it denote by τ) of the domain of σ is for itself in the domain of σ then the trace is finite and

$$\sigma(\tau)(s^o_m) \in D_{s^o_m} \text{ for all } m \in \{1,\ldots,n_o\})$$

[3] By $\uplus$ we denote the disjoint union. So we can assign to each component of the type a unique observable feature.

And finally, for the special symbol $\perp$ is $a(\perp)(n) = \perp_{\mathcal{F}}$ for all $n \in \mathbb{N}$. Together with a variable assignment v an intension a can be extended on terms in the usual way.[4] A variable assignment v for a PQ-model $\mathcal{F}$ for a given signature Σ is a total function which assigns to each variable $x \in X_s$ $(s \in S)$ a total function $v(x) : \mathbb{N} \to D_s$, where $v(x)(t) = v(x)(t')$ for all $t, t' \in \mathbb{N}$, to each program variable $x_p \in XP_s$ $(s \in S)$ a total function $v(x_p) : \mathbb{N} \to D_s$, and to each channel variable $x_c \in XC_s$ $(s \in S)$ a total function $v(x_c) : \mathbb{N} \to D_s \cup \perp_{\mathcal{F}}$. The set of all variable assignments associated to $\mathcal{F}$ is denoted by $Val(\Sigma, \mathcal{F}, X, XP, XC)$. A substitution v_x^b with an object b from the corresponding domain of quantification is defined by the following law: For all $s \in S$, $n \in \mathbb{N}$, and $x \in X_s \cup XP_s \cup XC_s$

$$v_x^b(y)(n) := \begin{cases} b & \text{if } y = x,\ x \in X_s,\ \text{and } b \in Q_{X_s} \\ b(n) & \text{if } y = x,\ x \in XP_s,\ \text{and } b \in Q_{XP_s} \\ b(n) & \text{if } y = x,\ x \in XC_s,\ \text{and } b \in Q_{XC_s} \\ v(y)(n) & \text{else} \end{cases}$$

We remind: The model of a process in a world $t \in \mathbb{N}$ is a set of observation traces (possible runs) starting in that world. Since each trace σ has a certain length $L(\sigma)$ the l.u.b. of $\{d(t,t')|t,t'$ is in the domain of $\sigma\}$, the view of a trace is in particular associated with an interval of time starting in that world. So, moments are not the only view of time. We have to consider intervals of time as contextual information for the accessibility relation $(\leq)$. So the accessibility relation will look more like an accessibility relation of an interval based temporal logic like [12] or [4]. Since our extensions are naturally assigned to worlds and not to intervals we will not change our logic to an interval temporal logic although there are several features in common.

4 A Language for Specifying Real-Time Systems

In this section we give an overview of well-formed formulas and how they are interpreted in a PQ-model. This provides the basis for a definition of a specification language for real-time systems.

In the section above we have mentioned sorts, function symbols, predicate symbols (rigid or not), process symbols , and the symbol $\perp$, together with logical, program, and channel variables as primitives of temporal signatures. On these primitives we build up two kinds of terms:

data terms: consisting of function symbols, logical variables, and program variables in a sort respecting manner.

channel terms: consisting of channel variables and the symbol $\perp$. A channel term does not appear in any other term.

Since we have restricted the use of special communication symbols and channel terms in the construction of terms, it is possible to stay on total functions as

[4] A detailed explanation of terms and why we can stay on total functions as intension on function symbols is given in the next section.

intension — nowhere appears $\perp_{\mathcal{F}}$ during the evaluation of a data term. Furthermore we assume for each signature Σ that it contains at least two sorts `time` and `boolean` together with special function and predicate symbols. Both `time` and `boolean` have a fixed semantical domain in each PQ-model. For $D_{\texttt{time}}$ we use $\mathbb{N} \cup \{\infty\}$ and the special function and predicate symbol 0_t denotes the g.l.b. of $\mathbb{N} \cup \{\infty\}$, ∞_t the l.u.b., -1 the predecessor, $+1$ the successor, and $<$ denotes less. Since our logic is linear we can use terms of the sort `time` for both worlds and measures. For $D_{\texttt{boolean}}$ we use $\{true, false\}$ and the two null-ary function symbols `t` and `f` to denote *true* and *false* respectively. We will call terms with standard extensions in $D_{\texttt{time}}$ as terms of sort `time`, and terms with standard extensions in $D_{\texttt{boolean}}$ as terms of sort `boolean` respectively.

Now we consider the construction of formulas, starting with two different primitives. Firstly, **atomic formulas** containing predicates over data terms, and equations of arbitrary terms as usual. Secondly, **command formulas** constructed from process symbols, data terms, and channel variables, the least set containing:

- *elementary commands*,
$$q(\mathbf{u} : \mathbf{c} : \mathbf{d} : \mathbf{u}')$$
a process symbol q together with four sequences of terms respecting the type of q. Two sequences of data terms $\mathbf{u}$ to represent the input values and $\mathbf{u}'$ for the output values, and two sequences of channel terms $\mathbf{c}$ for receiving (*in-channels*) and $\mathbf{d}$ for sending (*out-channels*).
- *sequential commands*, *parallel commands*, and *conditionals*,
$$(\alpha ; \beta) \ , \ (\alpha || \beta) \ , \text{and} \ \ (\textbf{if} \ \ b \ \alpha \ \beta)$$
composed commands where α and β are arbitrary commands while b is a term of sort `boolean`.

Other commands like **send** or **receive** which are often used as elementary commands should be defined by the user. So they can be specified in a problem specific manner. From these primitives — atomic and command formulas — the formulas of the linear temporal logic of processes were built up by adding the logical operators $\neg$ (not) and $\rightarrow$ (implication), as well as $\exists$, the concept of existential quantification for each kind of variables, and different restricted modal operators to deal with real-time aspects. We use

$$\Diamond \ , \ \Diamond_R$$

to denote the existence of a moment in time. For intervals in time we use

$$\blacklozenge \ , \ \blacklozenge_R \ , \ \blacklozenge^R \ , \text{and} \ \ \blacklozenge_R^{R'}$$

respectively. In both cases the subscript restricts the moment in time, while the superscript fixes the interval and so the additional contextual information for the

accessibility relation. A restriction R or R' is always a fragment of an atomic formula which has a wild card (?) for a term of sort `time`.

$$? = t$$
or
$$p(u_1, \ldots, u_{i-1}, \ ? \ , u_{i+1}, \ldots, u_n)$$

The idea is that we restrict the possible worlds and intervals to those which equals to the extension of t in the case of the equation or to those which are related to the extensions of the given terms u_k $(k \in \{1, \ldots, n\} \setminus \{i\})$ by the extensions of the n-ary predicate symbol p. While for the equation it is necessary that t is a term of sort `time`, a term u_k in the predicate formula only depends on the type of the predicate. We only have to require that the position of the wild card is normally occupied by a term of sort `time`. If the position for the wild card is obvious it can be omitted. So we can write for a predicate that is used in an infix notation e.g. $\leq$

$$\Diamond_{\leq t} \quad \text{or} \quad \Diamond_{t\leq}$$

to denote all moments that are less or greater then t respectively. We denote for a given signature Σ the set of all linear process formulas by $LPF(\Sigma)$.

We conclude this section with the definition of validity of a formula $\varphi \in LPF(\Sigma)$ in a given PQ-model $\mathcal{F}$ for Σ. According to the discussion above we have to regard moments in time w as well as intervals ℓ for the contextual information. By v we denote the variable assignment. We write

$$\mathcal{F}^{\ell}_{v,w} \models \varphi$$

to denote that φ is valid under the variable assignment v in the world w w.r.t the interval ℓ. For formulas where modal operators don't appear at the outermost position intervals aren't important. We can define

$$\mathcal{F}^{\ell}_{v,w} \models \mathbf{true}\ , \ \ \mathcal{F}^{\ell}_{v,w} \models (u_1 = u_2) \text{ iff. } u_1^{\mathcal{F}_{v,w}} = u_2^{\mathcal{F}_{v,w}}\ , \ \ \mathcal{F}^{\ell}_{v,w} \models \neg\varphi \text{ iff. } \mathcal{F}^{\ell}_{v,w} \not\models \varphi$$

and so on.[5] For command formulas or formula where modal operators appear at the outermost position the validity also depends on the contextual information given by the length (ℓ) of the interval.

$\mathcal{F}^{\ell}_{v,w} \models \Diamond_{p(u_1,\ldots,?,\ldots,u_n)}\varphi$ iff. there exists a $k \in \mathbb{N}$ where $k \leq \ell$ such that $\mathcal{F}^{\ell-k}_{v,(w+k)} \models \varphi$ and $(u_1^{\mathcal{F}_{v,w}}, \ldots, k, \ldots, u_n^{\mathcal{F}_{v,w}}) \in a(p)(w)$

$\mathcal{F}^{\ell}_{v,w} \models \blacklozenge^{p_2(r_1,\ldots,?,\ldots,r_m)}_{p_1(u_1,\ldots,?,\ldots,u_n)}\varphi$ iff. there exists a $k \in \mathbb{N}$ and a $l \in \mathbb{N} \cup \{\infty\}$ where $k + l \leq \ell$ such that $\mathcal{F}^{l}_{v,(w+k)} \models \varphi$ and $(u_1^{\mathcal{F}_{v,w}}, \ldots, k, \ldots, u_n^{\mathcal{F}_{v,w}}) \in a(p_1)(w)$ and $(r_1^{\mathcal{F}_{v,w}}, \ldots, k, \ldots, r_m^{\mathcal{F}_{v,w}}) \in a(p_2)(w)$

Now we will take a deeper look at commands.

[5] By $u_1^{\mathcal{F}_{v,w}}$ we denote the extension of the term u_1 in the world w.

$\mathcal{F}^{\ell}_{v,w} \models q((\ldots,u_i,\ldots):(\ldots,c_n,\ldots):(\ldots,d_m,\ldots):(\ldots,u'_k,\ldots))$ iff. there exists a trace $\sigma \in a(q)(w)$ with length $L(\sigma)=\ell$ and for each $w' \in \mathbb{N} \cap [w, w+\ell]$ it holds that $\sigma(w)(s^i_j) = u_i^{\mathcal{F}_{v,w}}$, $\sigma(w')(s^r_n) = c_n^{\mathcal{F}_{v,w'}}$, $\sigma(w')(s^s_m) = d_m^{\mathcal{F}_{v,w'}}$, and $\sigma(w+\ell)(s^o_m) = {u'}_k^{\mathcal{F}_{v,w+\ell}}$ if $w+\ell < \infty$.

We remember (see page 7) that the extension of a process symbol is a set of observational traces. So we had to verify that the terms in the argument positions match with one such possible trace. The first part of the argument, are the input terms which initially influence the process. These terms should match in the first world (w), while the terms for the communication (only channel variables or the symbol $\perp$) in the second and third part of the argument should match during the whole run (w'). Finally the output terms, the last part of the argument should only match if the observation trace is finite. In all these cases the corresponding value to a term is determined by it's sort and it's position in the argument vector, according to the equation ($*$) on page 7.

$\mathcal{F}^{\ell}_{v,w} \models (\alpha;\beta)$ iff. there exists a $l \in \mathbb{N}$ $l \leq \ell$, such that $\mathcal{F}^{l}_{v,w} \models \alpha$ and $\mathcal{F}^{\ell-l}_{v,w+l} \models \beta$ or $\ell = \infty$ and $\mathcal{F}^{\ell}_{v,w} \models \alpha$. Furthermore we need some variable conditions. Any program variable (x_p) which appears in α or β but not in the output part of any elementary command in α or β will stay unchanged i.e. $a(y_p)(w') = a(y_p)(w)$ for all $w' \in [w, w+\ell] \cup \mathbb{N}$. This is necessary to propagate input and output parameters over chains of commands, since program variables aren't rigid. We have similar conditions for the single subprocesses. The overall aim is that variables which are used in a subprocess but not influenced don't change during this subprocess.[6] We also have restrictions for channel variables which model communication. A channel variable which appears only in one subprocess (α or β) as a sending channel equals $\perp$ in the other.[7] For channel variables which appear in the receiving part no constraints are needed. In this process model the channel is dominated by the sending process. Synchronization in the sense of a minimal wait doesn't take place.

Since terms are used and not only variables we have the opportunity to write shorter formulas using matching facilities:

$$\exists x\ proc1(\ldots:\ldots:\ldots:f(x));proc2(x:\ldots:\ldots:\ldots)$$

$proc1$ will determine some x which fits through $f(x)$ in an observation trace of $proc1$'s extension. Then these x are propagated to the second process $proc2$.

$\mathcal{F}^{\ell}_{v,w} \models (\alpha||\beta)$ iff. there exists $l_1, l_2 \leq \ell$, such that $\mathcal{F}^{l_1}_{v,w} \models \alpha$ and $\mathcal{F}^{l_2}_{v,w} \models \beta$ and $\ell \in \{l_1\} \cup \{l_2\}$. As above there are restrictions on program and channel variables. Variables which appear in the earlier finished process don't change their values during the period where the subprocess but not the whole parallel process is finished. In the case of channel variables used for sending we make the same

[6] This is comparable with a closed world assumption or a kind of frame axiom.

[7] The channel variable appears in the sending part of an elementary command in this subprocess

restriction but we require that their value equals $\perp$.

$\mathcal{F}^{\ell}_{v,w} \models$ (**if** $b\ \alpha\ \beta$) iff. Firstly, $b^{\mathcal{F}_{v,w}} = true$ and $\mathcal{F}^{\ell}_{v,w} \models \alpha$. In this case we require that all variables which are used in (**if** $b\ \alpha\ \beta$) but not influenced by α don't change during the time interval $[w, w+\ell]\cup\mathbb{N}$. Similarly we require for all channels to which only β can write that they equal $\perp$. Or secondly, if $b^{\mathcal{F}_{v,w}} = false$ and $\mathcal{F}^{\ell}_{v,w} \models \beta$ we use the same restrictions as above by switching between α and β.

Now we declare a formula $\varphi \in LPF(\Sigma)$ to be valid in $\mathcal{F}$ at a moment w respecting the interval ℓ (denoted by $\mathcal{F}^{\ell}_{w} \models \varphi$) iff. $\mathcal{F}^{\ell}_{v,w} \models \varphi$ for each variable assignment v. Then a formula $\varphi \in LPF(\Sigma)$ is valid in $\mathcal{F}$ (denoted by $\mathcal{F} \models \varphi$) iff. φ is valid in the initial world 0 and in the initial interval ∞ ($\mathcal{F}^{\infty}_{0} \models \varphi$). And finally a formula $\varphi \in LPF(\Sigma)$ is valid ($\models \varphi$) iff. $\mathcal{F} \models \varphi$ holds for each $\mathcal{F}$.

5 Towards a Calculus

In the sections above we have introduced a temporal (modal) logic with an explicit process concept. Now we will sketch a calculus for doing proofs based on signed formulas. Modal calculi with signed formulas can be found for example in [5]. The idea is that by the signing we make interval borders explicit. Since our logic is quantified, and there a terms of various sorts especially `time` with an fixed interpretation, we can use the latter to sign our formulas, instead of using a new class of symbols. So we get the advantage that we can use the same mechanism for proving constraints and arbitrary formulas.

Definition 3 signed formulas. The set $SPF(\Sigma)$ of all signed process formulas of a signature Σ is the smallest set containing: $(t : t' : \varphi)$, $\neg\phi$, $\phi \to \psi$, and $\exists x\ \phi$ for all formulas $\varphi \in LPF(\Sigma)$, rigid terms t and t' of sort `time`, signed formulas ϕ and ψ, and logical, or program, or channel variables x.

The idea is, that the terms t and t' denote lower and upper bounds of the current interval, in which the formula should be evaluated. So, in an easy way one defines validity of a signed formula in a PQ-model $\mathcal{F}$ under a variable assignment v:
If $a(t)(0) \in \mathbb{N}, a(t')(0) \in \mathbb{N} \cup \{\infty\}$, and $a(t)(0) \leq a(t')(0)$ then[8]

$$\mathcal{F}_v \models (t : t' : \varphi) \ \text{ iff. } \ \mathcal{F}^{a(t')(0)-a(t)(0)}_{v,a(t)(0)} \models \varphi$$

else

$$\mathcal{F}_v \not\models (t : t' : \varphi)$$

For composed signed formulas this definition can be extended in the usual way.

The connection between formulas and signed formulas is then established by the following theorem.

Theorem 4 explicit intervals. *Let* $\varphi \in LPF(\Sigma)$ *then:*[9]

$$\mathcal{F} \models \varphi \ \textit{ iff. } \ \mathcal{F} \models (0_t : \infty_t : \varphi)$$

[8] In this case we have a proper interval.

[9] Both 0_t and ∞_t are rigid null-ary function symbols of sort time with the standard extensions 0 and ∞.

This opens up the possibility to develop a calculus on the basis of signed formulas. Since only an interactive proof system seems to be desirable, because of the expressiveness and therefore incompleteness of our logic, we have decided to develop a sequent calculus.[10] To give an insight into the calculus we present a small collection of rules.

$$\frac{\Gamma, (t{:}t'{:}\varphi) \Rightarrow \Delta \quad \Gamma \Rightarrow (t{:}t'{:}true), \Delta}{\Gamma \Rightarrow (t{:}t'{:}\neg\varphi), \Delta} \qquad \frac{\Gamma \Rightarrow (t{:}t'{:}\varphi_1), \Delta \quad \Gamma, (t{:}t'{:}\varphi_2) \Rightarrow \Delta}{\Gamma, (t{:}t'{:}\varphi_1 \rightarrow \varphi_2) \Rightarrow \Delta}$$

$$\frac{\Gamma \Rightarrow \phi^{\tau}_{x}, \Delta \quad \Gamma \Rightarrow (0_t{:}\infty_t{:}\exists x_0\ \Box x_0 = \tau), \Delta}{\Gamma \Rightarrow \exists x\ \phi, \Delta} \qquad \frac{\Gamma, (t, t^{\alpha}{:}\alpha), (t{:}t^{\beta}{:}\beta), \text{var-conds} \Rightarrow \Delta}{\Gamma, (t{:}t'{:}\alpha||\beta) \Rightarrow \Delta}$$

The rule dealing with negation includes $(t : t' : true)$ for guaranteeing a proper interval. Only in this case the formulas $(t : t' : \neg\varphi)$ and $\neg(t : t' : \varphi)$ are equivalent. For the presented implication rule a proper interval is not necessary, since the implication is on the left hand side of the sequent. In the rule dealing with existential quantification x and x_0 are logical variables (x_0 new). So we have to guarantee that the term τ is rigid. In the rule with the parallel operator *var-conds* stands for a set of formulas guaranteeing the variable conditions mentioned in the semantic definition of the section before. To get a sound calculus we have proved the following lemma for the whole set of rules.

Lemma 5 local soundness. *Let* $R = \frac{prem_1 \cdots prem_n}{concl}$ *be a rule of then for all* $\mathcal{F}$ *holds: If* $\mathcal{F} \models prem_i$ $(1 \leq i \leq n)$ *then* $\mathcal{F} \models concl$.

We will conclude this section with the soundness theorem of the calculus.

Theorem 6 soundness. *2 For all closed formulas* $\varphi \in LPF(\Sigma)$ *holds:*

$$\textit{If} \quad \vdash (0_t : \infty_t : \varphi) \quad \textit{then} \quad \models \varphi$$

Proof: Application of lemma 5 and theorem 4.

6 Examples

The following examples are intended to illustrate the capabilities of logic and calculus. The first example deals with expressiveness and shows that term generated structures can be described.

Example 1 expressiveness. Let us assume a signature containing sorts `L` and `E` together with a null-ary function symbol *nil* of type () $\rightarrow$`L` and a binary function symbol *cons* of type (`E`,`L`)$\rightarrow$`L`.[11] Furthermore let l and x be logical variables and x_p be a program variable all of sort L, and e a logical variable of sort E. Then by

$$\forall l\ \exists x_p\ (\Diamond\ x_p = l) \wedge x_p = nil \wedge \Box\ (\exists x\ \exists e\ x = x_p \wedge \Diamond_{=0_t+1}\ x_p = cons(e, x))$$

[10] By conceptual domains we have expressive power of second-order arithmetic.
[11] The sorts `time` and `boolean` are always given implicitly.

we have specified `L` as the sort of all "lists" over `E`. For each list l exists a concept for a program variable (non-rigid) sometimes equal to l, which is equal to nil in the initial world and growing up from world to world by appending one element until it is equal to l. So each list can be denoted by $cons(e, cons(\ldots))$ or nil.[12]

The second example demonstrates the possibility to deal with real-time aspects. We describe a sending process $send$ where the time of transmitting a data element x over a channel c depends on the data element itself.

Example 2 time and data. Let us assume a signature containing a sort `S` and a process symbol $send$ of type (`S`::`S`:), and a function symbol f of type `S`$\rightarrow$ `time` then by

$$\forall t, x, c \; \blacksquare_{=t} \; (send(x :: c :) \rightarrow t = f(x))$$

the duration of $send$ is $f(x)$ time steps.

Finally we present a more elaborated example dealing with sending and receiving of lists.

Example 3 parallel communication. For this example the following signature is assumed containing sorts, functions, and process symbols.

sorts: `list`, `data_record`, `data` ;
functions: nil : () $\rightarrow$ `list`; `/* the empty list */`
$cons$: (`data`,`list`) $\rightarrow$ `list`;
rec : (`boolean`, `data`) $\rightarrow$ `data_record`; `/* a container */`
d : () $\rightarrow$ `data`; `/* some data */`
processes: sl : (`list` :: `data_record` :); `/* sends a list */`
sd : (`data` :: `data_record` :); `/* sends a single data */`
se : (:: `data_record` :); `/* signals the end of a list*/`
rl : (: `data_record` :: `list`); `/* receives a list */`
rd : (: `data_record` :: `data`); `/* receives a single data */`
re : (: `data_record` ::); `/* receives the end signal */`
logical variables: x : `data`; l, l_1 : `list`; t : `time`;
channel variables: c : `data_record`;

Firstly we describe the sending part.

$$\forall c \; \blacksquare(sl(nil :: c :) \rightarrow se(:: c :))$$
$$\forall x, l, c \; \blacksquare(sl(cons(x, l) :: c :) \rightarrow (sd(x :: c :); sl(l :: c :)))$$
$$\forall x, c, t \; \blacksquare_{=t}(sd(x :: c :) \rightarrow (t < \infty \wedge (\Box_{<t} c = \bot) \wedge (\Diamond_{=t} c = rec(\mathtt{t}, x))))$$
$$\forall c, t \; \blacksquare_{=t}(se(:: c :) \rightarrow (t < \infty \wedge (\Box c = \bot) \wedge (\Diamond_{=t} c = rec(\mathtt{f}, d))))$$

The process sl which sends a list depends on two primitive processes. The first is se which signals the end of the transmission by sending a data record containing the boolean symbol `f`, and the second is sd which sends a single data x encapsulated together with the boolean symbol `t` in the data record $rec(\mathtt{t}, x)$.

[12] The first order axioms which guarantee that the generation principal is indeed freely are omitted.

Both se and sd are doing nothing during a finite period of time t, until they send their information and terminate. In an analogous way we describe the process of receiving rl by primitives re and rd.

$$\forall c\ \blacksquare(rl(:c::nil) \to re(:c::))$$
$$\forall x,l,c\ \blacksquare(rl(:c::cons(x,l)) \to rd(:c::x); rl(:c::l))$$
$$\forall x,c,t\ \blacksquare_{=t}(rd(:c::x) \to ((\Box_{<t}c = \bot) \wedge (t < \infty \to \Diamond_{=t}c = rec(\mathtt{t},x))))$$
$$\forall c,t\ \blacksquare_{=t}(re(::c:) \to ((\Box_{<t}c = \bot) \wedge \wedge(t < \infty \to (\Diamond_{=t}c = rec(\mathtt{f},d)))))$$

With the assumption that `list` is generated by $cons$ and nil and that the container `data_record` is freely generated by rec it is possible to prove by induction that for all pairs of lists l and l_1, and for all channels (c) a "parallel execution" of the processes sl and rl implies that the values of l and l_1 are the same. This means the process rl has received the list l which was sent by sl.

$$\forall l,l_1,c\ \blacksquare((sl(l::c:)||rl(:c::l_1)) \to l = l_1)$$

We have proved the last example with machine support, by a prototypical implementation of the calculus in the GROOVY proof assistant (for a look and feel, see [13]), based on a hyper graph method developed by Roland Preiß.

The main idea of the proof is structural induction over lists. So we prove that if nil is sent nil will be received. For this we use a lemma that denotes "if end is signaled no data can be received". For the induction step we have to consider the transmission of a single data. For this we prove that sending is finite and the duration equals the receiving, and both the received and the sent data are the same. This is done by further lemmas which guarantee that the end signal can't be received if a single data item is sent, and that the receiving process doesn't wait infinitely. So the calculus allows us to do proofs in a natural way. Although we only have a prototypical implementation where each rule is applied interactively, the proof was done in some hours after fixing the intermediate lemmas, and so was the proof idea.

7 Conclusions and Future Research

We have discussed a method for a direct process embedding in structures for quantified temporal logic based on specific semantical objects, called process behavior sets, which are used as extensions for process symbols. So we have established a basis for proof methods to take advantage of the structural information located in process descriptions.

Currently we are evaluating these concepts on larger examples. So we want to detect more elaborated rules that respect the available structural (syntactical) information of the specified processes. The endeavor is to develop a proof method based on something like symbolic execution and induction (compare [8]), a technique which has been successfully applied in the area of sequential systems. We are also working on the evaluation of the embedding technique in other temporal structures (e.g. branching time) and other observable features. By the latter we want to get more realistic process concepts.

Furthermore we are working on a sub-language to describe "executable" specifications and refinement steps. The objective is to get a specification technique and language which covers the entire development process.

Acknowledgment

I am indebted to W. Menzel for valuable comments and W. Ahrendt, E. Habermalz, R. Preiß, and A. Schönegge for discussions on the topic of the paper.

References

1. R. Alur and D. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Proc ICALP 90: Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
2. R. Alur and Th.A. Henzinger. Logics and models of real time: A survey. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proc. Rex Workshop Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
3. J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In A. Ponse, C. Verhoef, and S.F.M. Vlijmen, editors, *Algebra of Communicating Processes*, pages 1–25. Utrecht 1994, Workshop in Computing, Springer-Verlag, 1995.
4. Z. Chaochen. Duration calculi: An overview. In D. Bjørner, M. Broy, and I.V. Potossin, editors, *Formal Methods in Programming and their Application*, volume 735 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.
5. M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel publishing company, Dordrecht Boston Lancaster, 1983.
6. Th. Fuchß, W. Reif, G. Schellhorn, and K. Stenzel. Three selected case studies in verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software – Final Report*, volume 1009 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.
7. J.W. Garson. Quantification in modal logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, chapter 5, pages 249–307. D. Reidel publishing company, 1984.
8. M. Heisel, W. Reif, and W. Stephan. Program verification by symbolic execution and induction. In K. Morik, editor, *Proc. 11th German Workshop on Artificial Intelligence*, number 152 in Informatik Fachberichte. Springer-Verlag, 1987.
9. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
10. A. Mazurkiewicz. Basic notions of trace theory. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Artificial Intelligence*. Springer, June 1988.
11. R. Milner. *Calculus of communicating systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
12. B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE computer*, pages 10–19, February 1985.

13. R. Preiß. GROOVY: a Graphical Proof and Visualization System — home page. URL: http://i11www.ira.uka.de/~groovy/.
14. W. Reisig. *Petri nets: an introduction*. Springer-Verlag, 1985.
15. G. Winskel. An introduction to event structures. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Artificial Intelligence*, pages 364–397. Springer-Verlag, June 1988.
16. G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4 Semantic Modelling, chapter 1, pages 1–148. D. Reidel publishing company, 1995.